Aalto University
School of Science

Coordination Models and Techniques for Big Data and Machine Learning Systems

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, https://rdsea.github.io

Learning objectives

- **Analyze the role of coordination techniques, their complexity and diversity in big data/ML systems**
- **Understand and apply orchestration models, common tools and design patterns**
- **Understand and apply choreography models, common tools and design patterns**
- **Understand, define and develop ML Model Serving**

Coordination complexity and diversity

Examples of common tasks

Discussion:
- **ML phases & tasks**
- **Software stack**
- **Execution environments**
  - Computing resources
- **R3E**

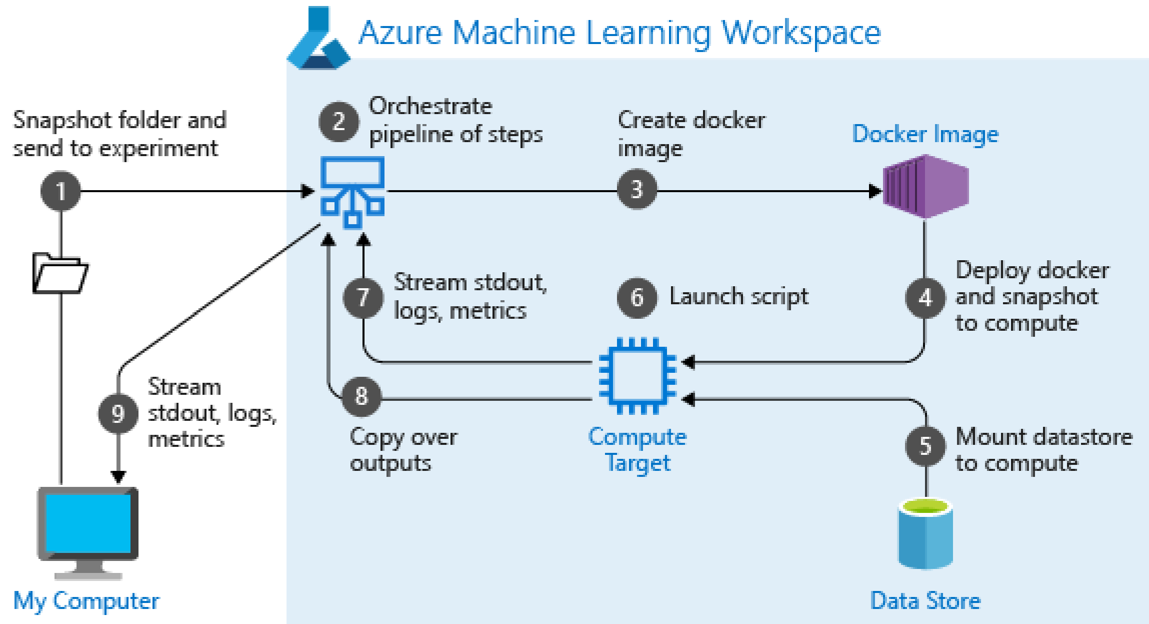
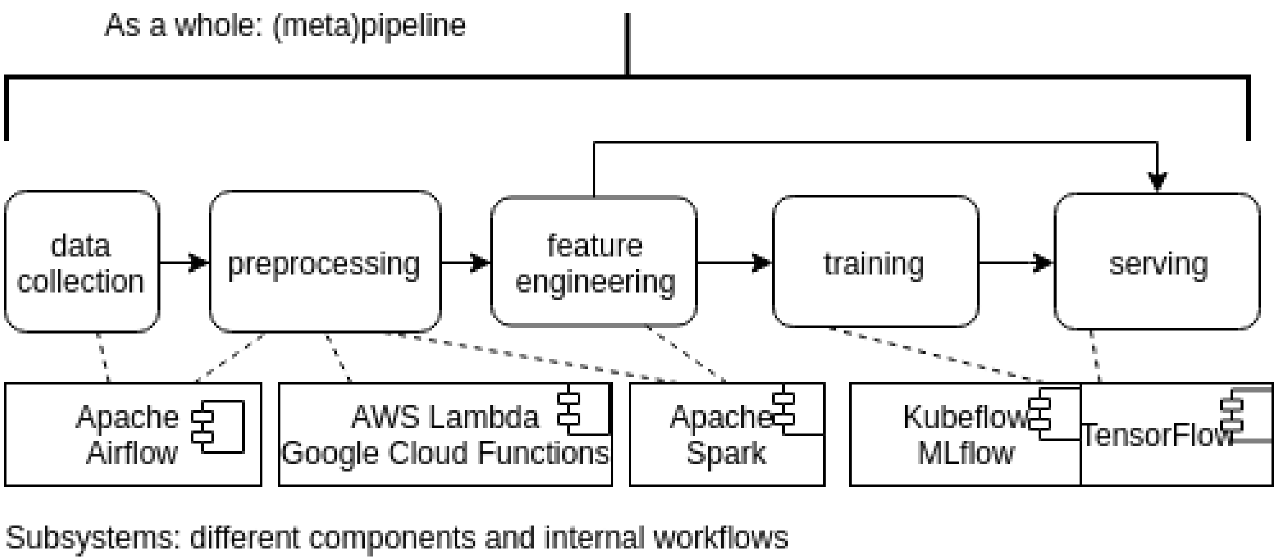


Figure source: https://docs.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines

Recall: big data/ML systems

- **Multiple levels:**
  - Meta-workflow or -pipeline
  - Inside each phase: pipeline/workflow or other types of programs

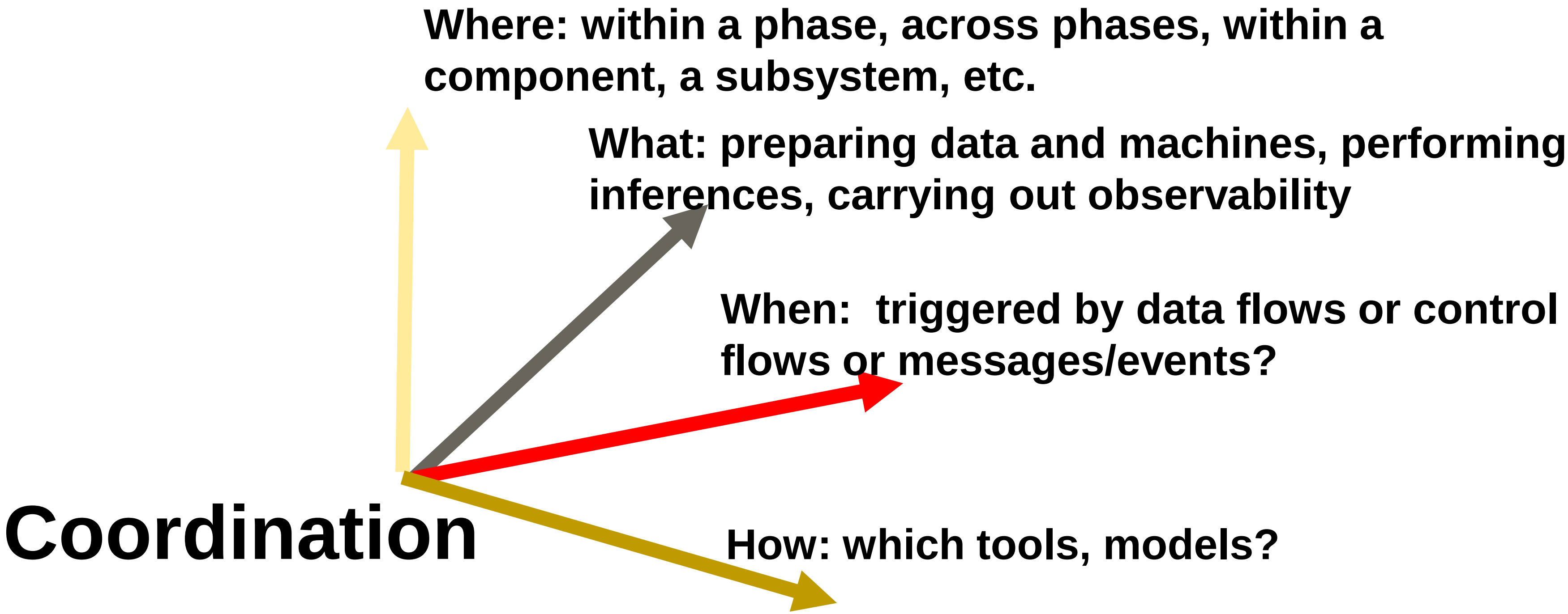


Automation vs manual tasks: share your current way?

Main issues related to coordination

- **How to coordinate phases and tasks in big data/ML systems**
  - automation is an important requirement, why?
- **How to prepare artefacts and resources for big data/ML systems**
- **How to manage tests and experiments**
  - trial computing configurations, inputs/results collection
- **How to control for assuring R3E for the pipeline execution**
  - end-to-end R3E requires coordination
  - issues in internal and external services

W3H: what, when, where and how for coordination



Diversity and Complexity

- **Diversity**
  - so many tools/frameworks in a single big data/ML system
    → *a single coordination model/tool might not be enough*
  - there exist many coordination systems (included your specific implementation)
    → *which ones should we select?*
- **Complexity, due to**
  - integration models with big data/ML components and infrastructures
  - very large-scale
  - runtime management: performance, failures, states

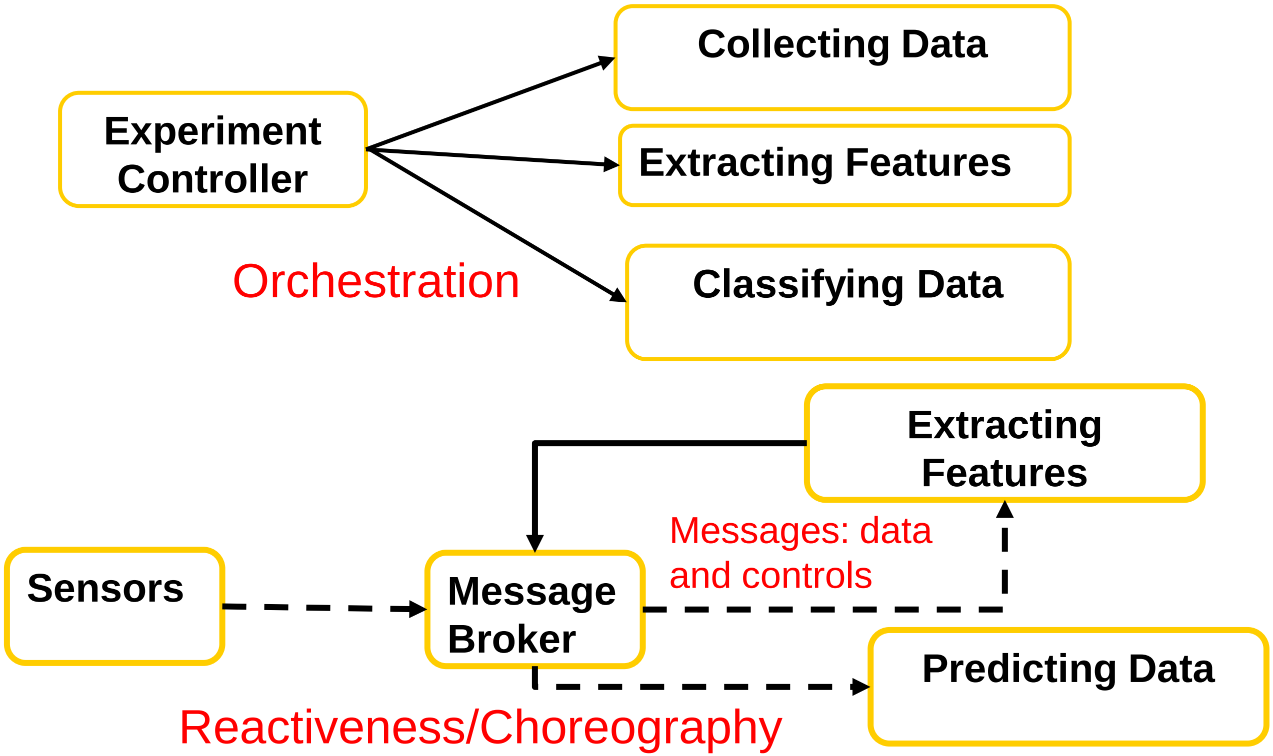
Key notions

- **Workflow and Task/Activity/Step abstraction**
  - a task can encapsulate a “complex workflow”
- **Big Data/ML software frameworks**
  - for implementing big data/ML capabilities
- **Platform services for coordination**
  - services offering features/functionality for executing “tasks”
  - single or multiple providers?
- **Execution environments and resources**
  - single platform or cross (heterogeneous) platforms

Coordination styles

- **Coordination models for Big Data/ML systems**
  - orchestration and reactiveness/choreography
- **Orchestration**
  - task graphs and dependencies are based on control or data flows
  - dedicated orchestrator: tasks triggered based on completeness of tasks or the availability of data
- **Reactiveness/choreography**
  - follow reactive model: tasks are reacted/triggered based on messages

Orchestration and Reactiveness



System issues impacting coordination

- **Main situations:**
  - within the same system/infrastructure
    - *all services and computing resources belong to the same platform/infrastructure*
    - *e.g., running everything with Google Cloud or Microsoft Azure*
  - across systems/infrastructures
    - *services in different clouds or cloud data centers*
    - *e.g., Edge-cloud infrastructures*
  - with the same software stack or not?
- **How such situations would affect the coordination?**

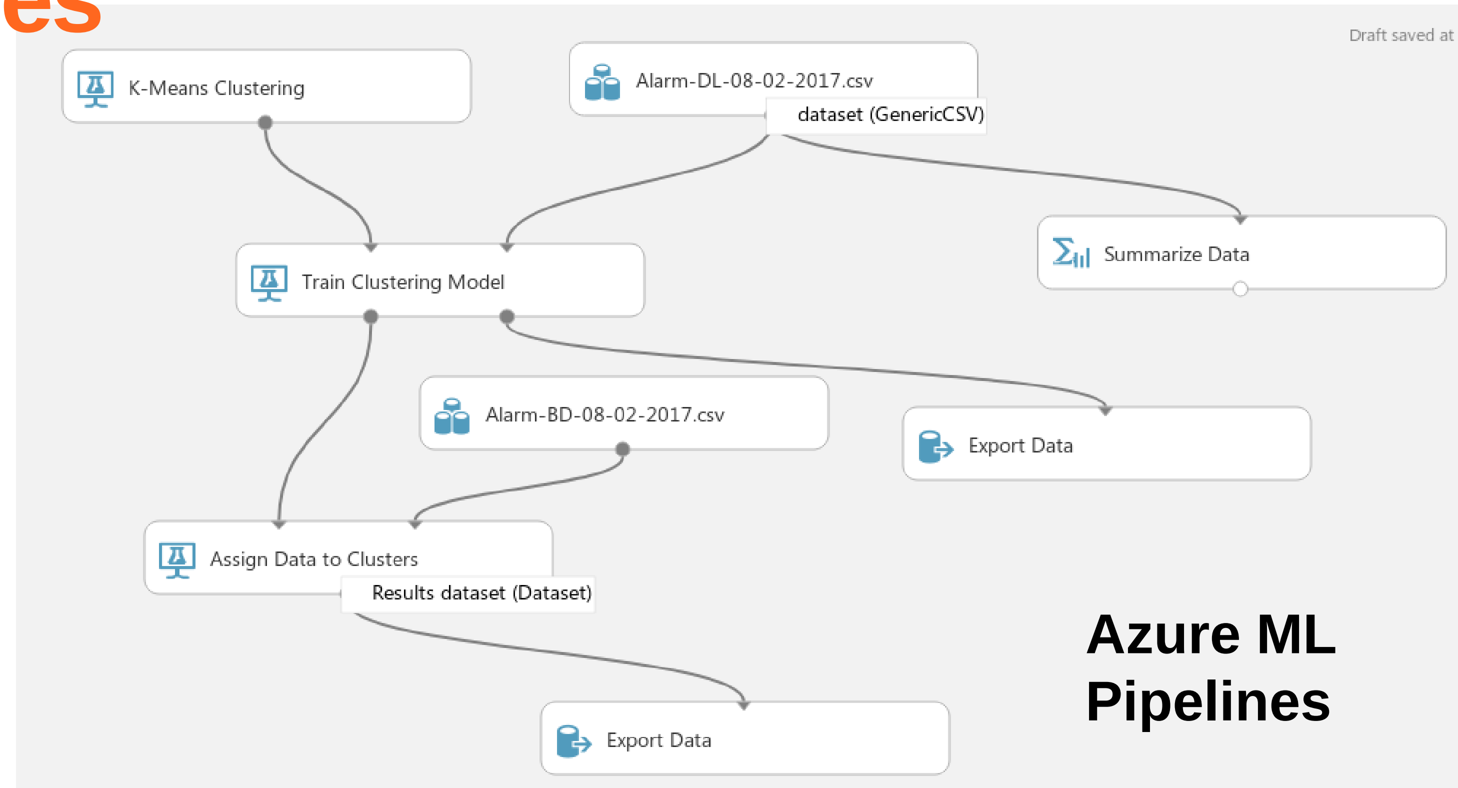
Coordination with workflow techniques

The orchestration style

Orchestration architectural style: design

- **Workflow architectures are known**
  - Big Data/ML systems: leverage many types of services and cloud technologies
- **Required components**
  - workflow/pipeline specifications/languages (also UI)
  - data and computing resource management
  - orchestration engines (with different types of schedulers)
- **Execution environments**
  - cloud platforms (e.g., VMs, containers, Kubernetes)
  - heterogenous computing resources (PC, servers, Raspberry PI, etc.)

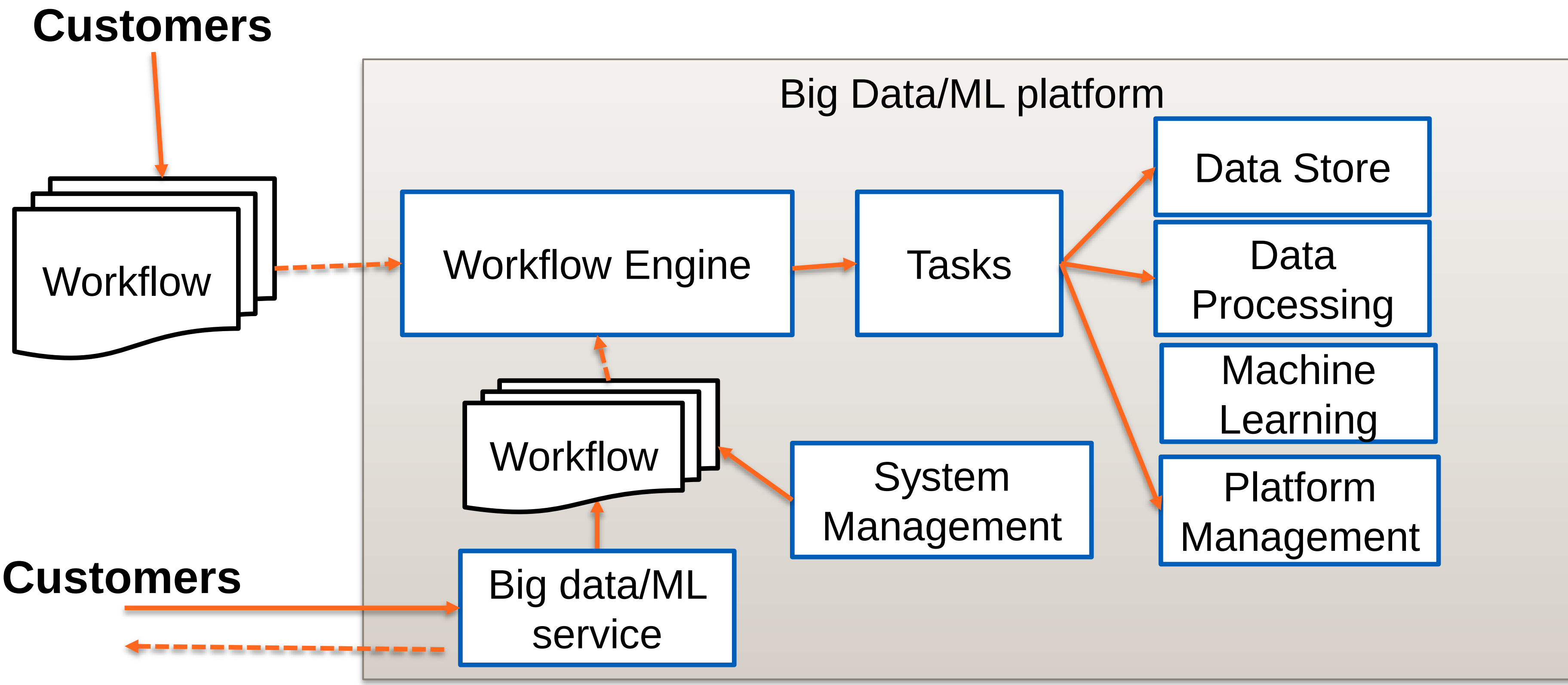
Example: workflow used in ML pipelines



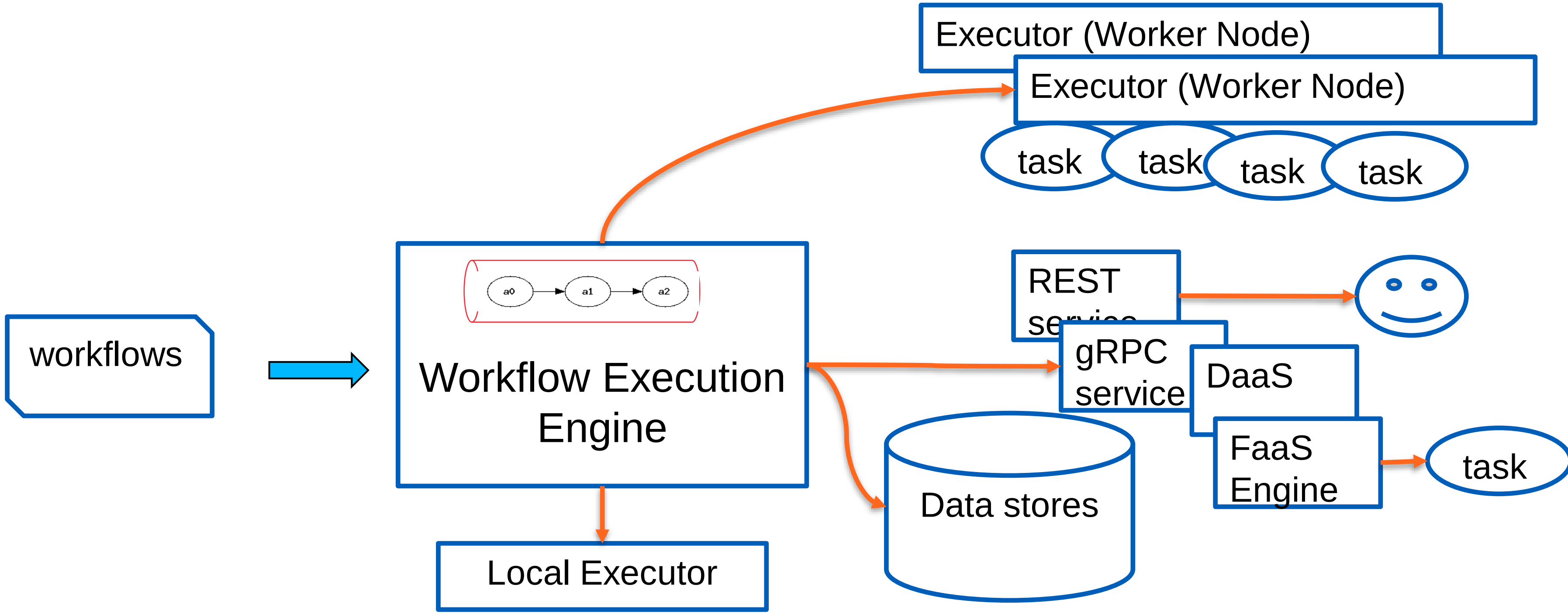
So what is behind the scene?

Azure ML Pipelines

Workflows in big data/ML systems



Common workflow execution models



Executors: containers, VMs, common OS processes

Key components

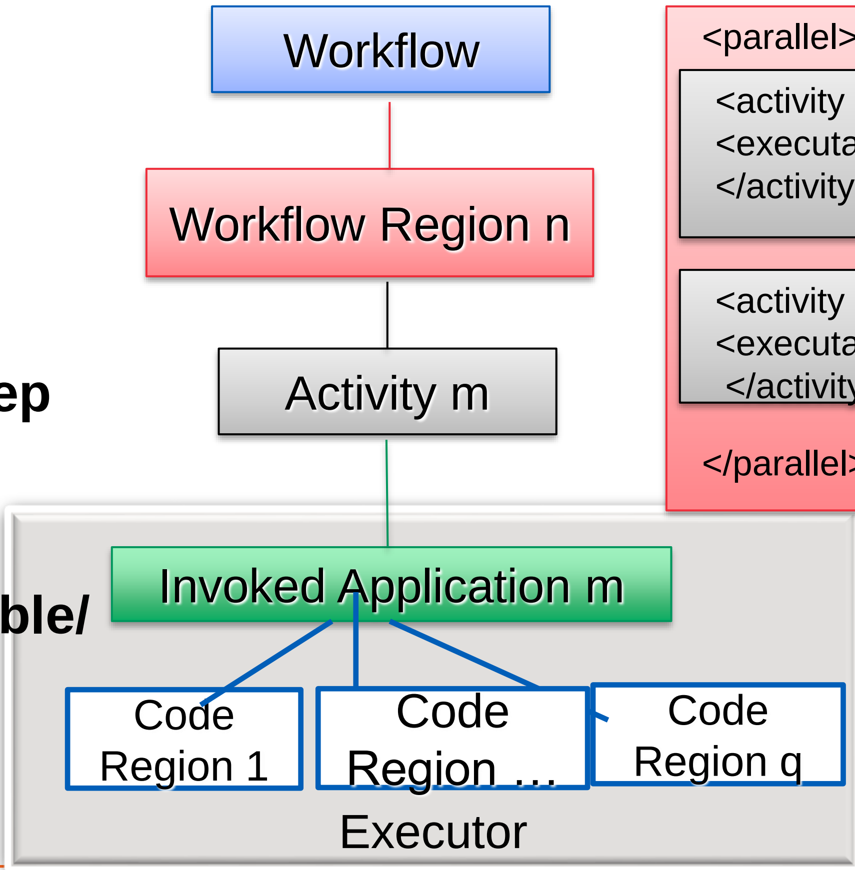
- **Tasks/Activities**
  - describe a single work (it does not mean small)
  - tasks can be carried out by humans, executables, scripts, batch applications, stream applications and services.
- **Workflow Languages**
  - how to structure/describe tasks, dataflows, and control flows
- **Workflow Engine**
  - execute the workflow by orchestrating tasks
  - usually call remote services to run tasks

Structured view of workflows

abstract example

Task/Step

Service /executable/ script




```
<parallel>
  <activity name="mTask1">
  <executable name="mTask1"/>
  </activity>

  <activity name="pTask2">
  <executable name="cbpprediction"/>
  </activity>
</parallel>
```

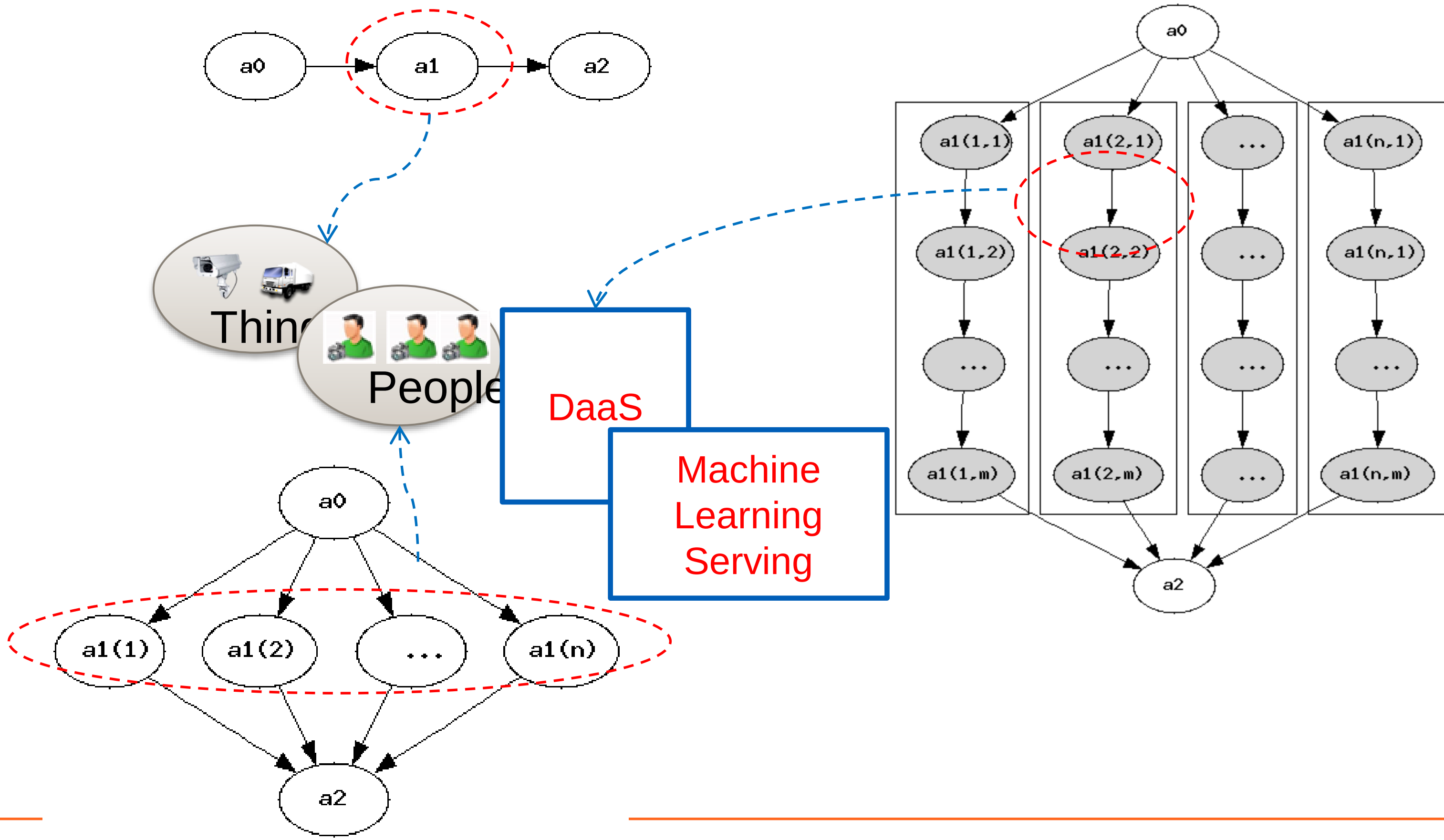
```
cbpprediction.py
def find_target_regression_model(dataset):
    selecteddata=selecteddata[TARGET_FEATURES].dropna()

    target_lin_reg =LinearRegression()
    target_lin_reg.fit(X,Y)
```

Structured view

- **Invoked applications within a task/activity**
  - can be a containerized service, script, python program, a function, etc.
  - can be designed for different purposes: e.g. computation, management or prediction
- **Encapsulating the whole workflow**
  - the whole workflow can be encapsulated with in a service
  - thus the whole workflow can be invoked via a service call for multiple consumers

Tasks orchestration



Runtime aspects

- **Parallel and distributed execution**
  - tasks are deployed and running in different machines
  - multiple workflows are running
- **Long running**
  - can be hours!
- **Checkpoint and recovery**
- **Monitoring and tracking**
  - which tasks are running, where are they?
- **Stateful management**
  - dependencies among tasks w.r.t control and data

Describing workflows

- **Programming languages with procedural code**
  - general- and specific-purpose programming languages, such as Java, Python, Swift
  - common ways in big data and ML platforms
- **Descriptive languages with declarative schemas**
  - BPEL, YAML, and several languages designed for specific workflow engines
  - common in business and scientific workflows
  - YAML is also popular for big data/ML workflows in native cloud environments

Workflow frameworks

- **Often running in the same infrastructure**
- **Task-driven or data-driven specification**
- **Generic workflows**
  - Use to implement different tasks, such as machine provisioning, service calls, data retrieval
    - *Examples: Airflow, Argo Workflows*
- **Specific workflows for specific purposes**
  - E.g., Kubeflow (https://github.com/kubeflow/pipelines)

Workflow frameworks

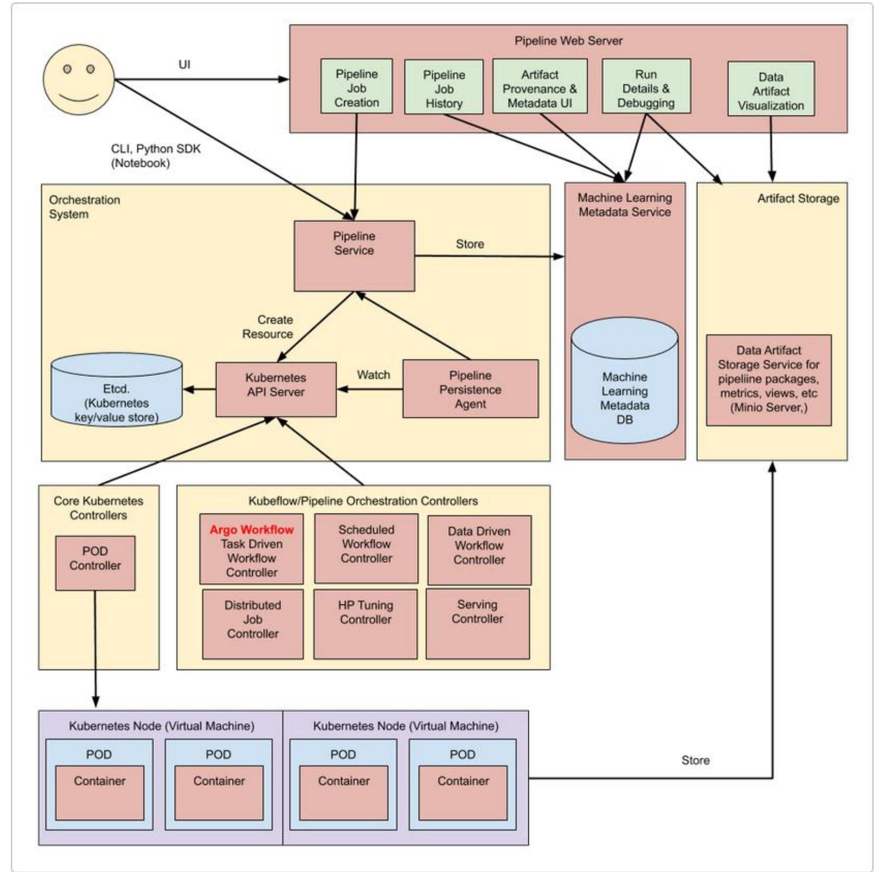
- **Apache Airflow**
  - https://airflow.apache.org/, also used by Google Composer
- **Argo**
  - https://argoproj.github.io/argo/, used in Kubeflow
- **Prefect**
  - https://www.prefect.io/
- **Uber Cadence**
  - https://github.com/uber/cadence

What purposes are for using workflows?

- **For coordinating big data/ML phases/stages in the pipeline**
- **For implementing tasks within a phase/stage**
  - implement data preprocessing task
  - implement training tasks
  - implement experiment management
  - batch model for machine learning serving
- **Which ones? No easy answer**
  - separate – as a framework/service
  - integrated within big data/ML frameworks

Examples: Kubeflow

- **End-to-end Orchestration**
- **Orchestration is based on workflows**
- **Using “Orchestration controllers”**

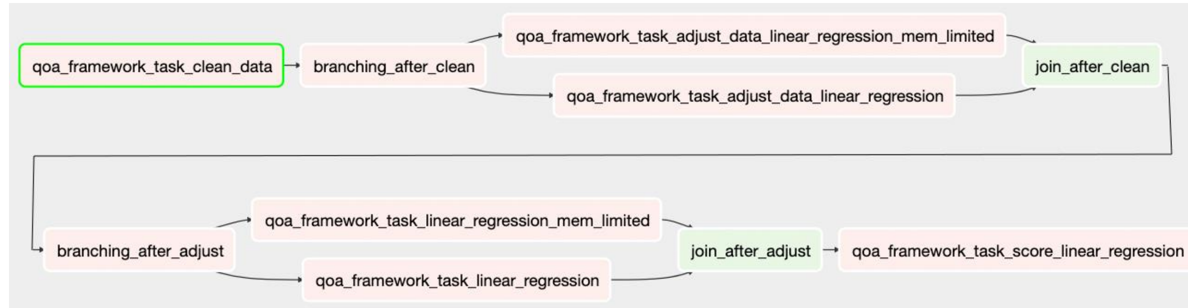


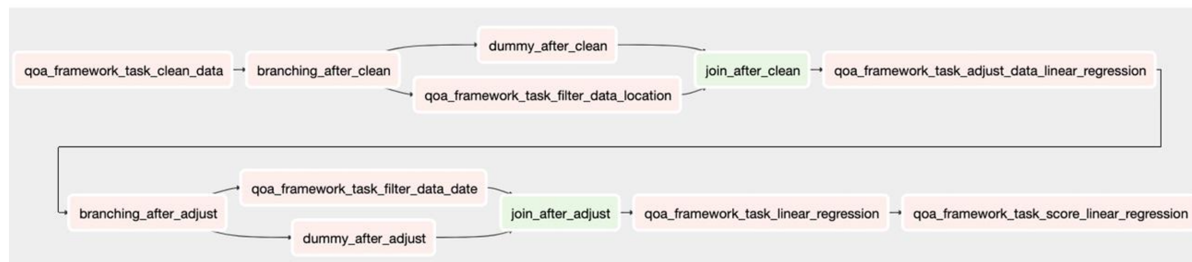
https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/

- Discussion: dealing R3E with ML workflows?
  - Where, What, When and How

Examples: Coordinating tasks

- Discussion: dealing R3E with ML workflows?
  - Where, What, When and How

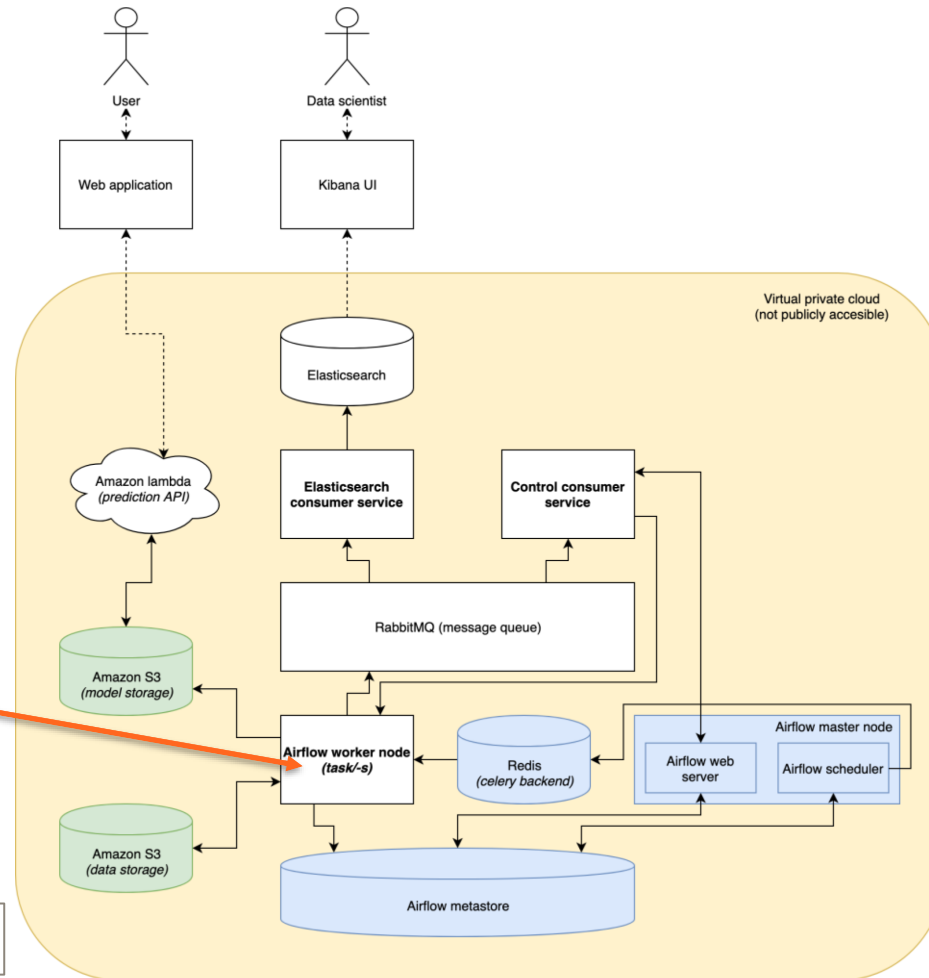




Source: Kreics Krists, „Quality of analytics management of data pipelines for retail forecasting,", Aalto CS Master thesis, 2019

Examples: Exchanging metrics for R3E coordination

Monitoring various metrics, including user-defined quality of data



Source: Kreics Krists, „Quality of analytics management of data pipelines for retail forecasting“, Aalto CS Master thesis, 2019

Example: Apache Beam

- **Goal: separate from data processing pipelines from backend execution engines**
  - Focus on pipeline design

Pipeline

Read data → Processing → …. → Store result

Flink

Apache Spark

Dataflow

Example: Apache Beam

- **https://beam.apache.org/**
- **Suitable for data analysis processes that can be divided into different independent tasks**
  - ETL (Extract, Transform and Load) & Data Integration
  - ML pipeline implementation
- **Execution principles:**
  - Mapping tasks in the pipeline to concrete tasks that are supported by the selected back-end engine
  - Coordinating task execution like workflows.

Example: Apache Beam Basic programming constructs

- **Pipeline:**
  - For creating a pipeline
- **PCollection**
  - Represent a distributed dataset
- **Transform**
  - [Output PCollection] = [Input PCollection] | [Transform]
  - Possible transforms: ParDo, GroupByKey, Combine, etc.
  - Partition: split the data

Example: Apache Beam

- **Data preprocessing and featuring engineering**
  - could also be for data validation
- **Preparation for training**
  - processing and partitioning data
- **Inferences**
  - implement inference functions which can be called within a pipeline, e.g. using ParDo()

Aalto University
School of Science

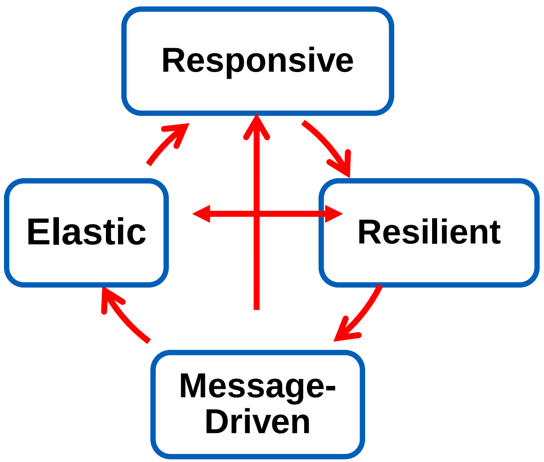
Coordination with messaging

The reactiveness/choreography

Choreography: Reactive systems for Big Data/ML

Do you remember key principles of reactive systems?

Reactive systems



Source: https://www.reactivemanifesto.org/

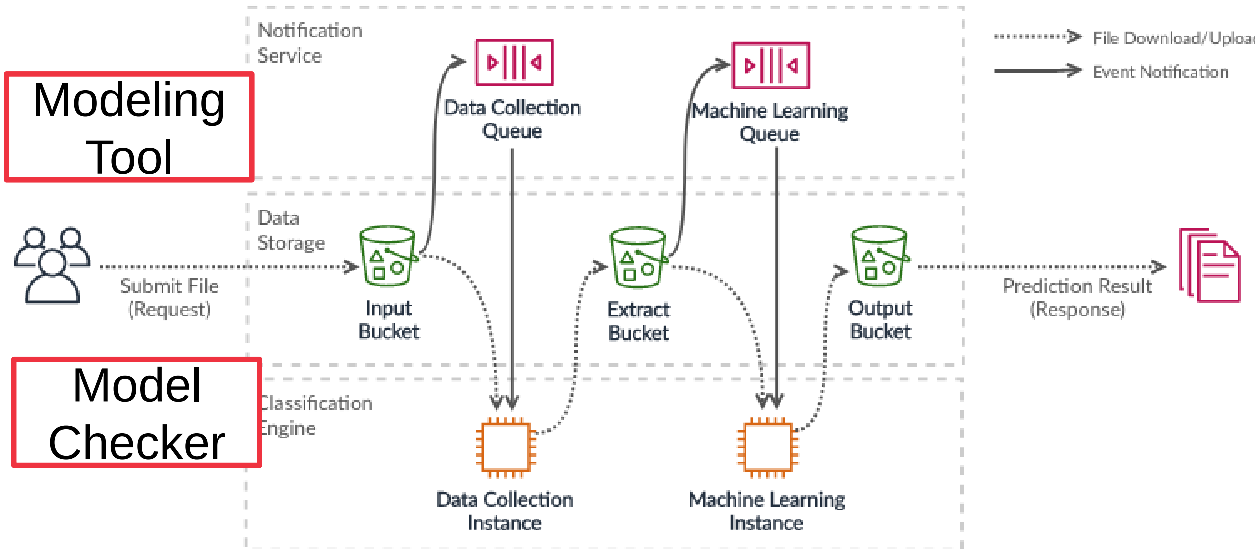
- Responsive: quality of services
- Resilient: deal within failures
- Elastic: deal with different workload and quality of analytics
- Message-driven: allow loosely coupling, isolation, asynchronous

Reactive systems for Big Data/ML: methods

- **Have different components as services**
  - components can come from different software stacks
  - components for doing computation as well as for data exchange
- **Elastic computing platforms**
  - platforms should be deployed on-demand in an easy way
- **Using messages to trigger tasks carried out by services**
  - messages for states and controls as well as for data
  - heavily relying on message brokers and lightweight triggers/controls (e.g., with serverless/function-as-a-service)

Examples: do-it-your-self ML classification for BIM

- Discussion: dealing R3E with ML workflows?
  - Where, What, When and How

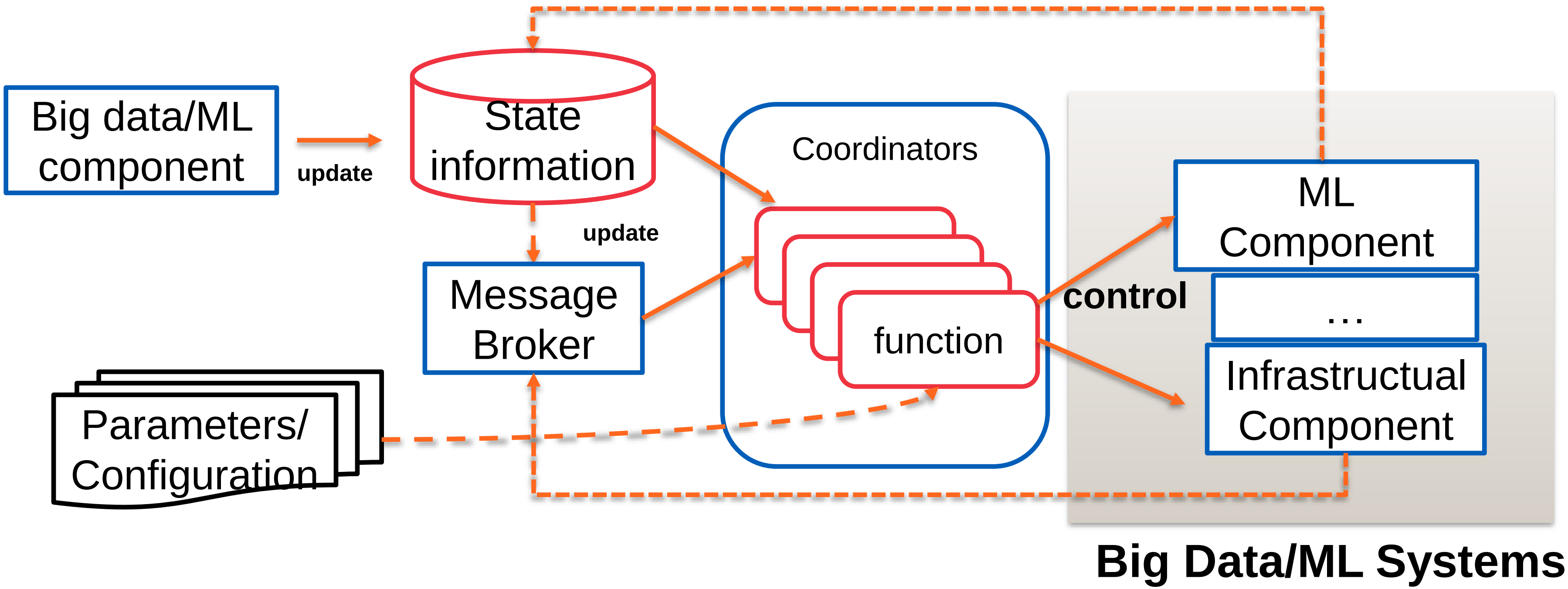


Source: Minjung Ryu, „Machine Learning-based Classification System for Building Information Models “, Aalto CS Master thesis, 2020

Which frameworks

- **Message brokers**
  - Kafka, RabbitMQ, Amazon SQS, …
  - types of messages and semantics must be defined clearly
- **Triggers and controls**
  - serverless/function-as-a-service can be used: trigger a function based on a message
    - *AWS Lambda, Google Cloud Function, Knative, Kubeless, OpenFaaS, Azure Functions*
    - *We are discussing to use serverless for "coordination"*
  - self-implementation of triggers listening messages

Common architecture



Example: Serverless for coordination

- **Training preparation**
  - before running a training: you move data from sources to stage, ship the code and prepare the environment
- **Coordination of ML phases**
  - do the coordination of three phases: data preprocessing, training and take the best model to deploy to a serving platform
- **Experiment results gathering**
  - you run experiments in different places. There are several logs of results, you gather them and put the result into a database

Serverless as functions within ML workflows

- **Tasks in ML can be implemented as a function**
- **Thus a workflow of functions can be used to implement ML pipelines**
  - Example: using serverless to implement data preprocessing

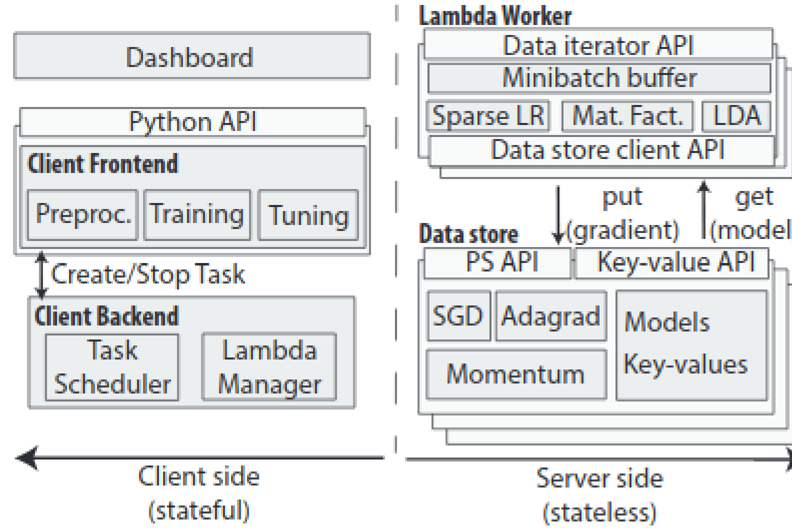


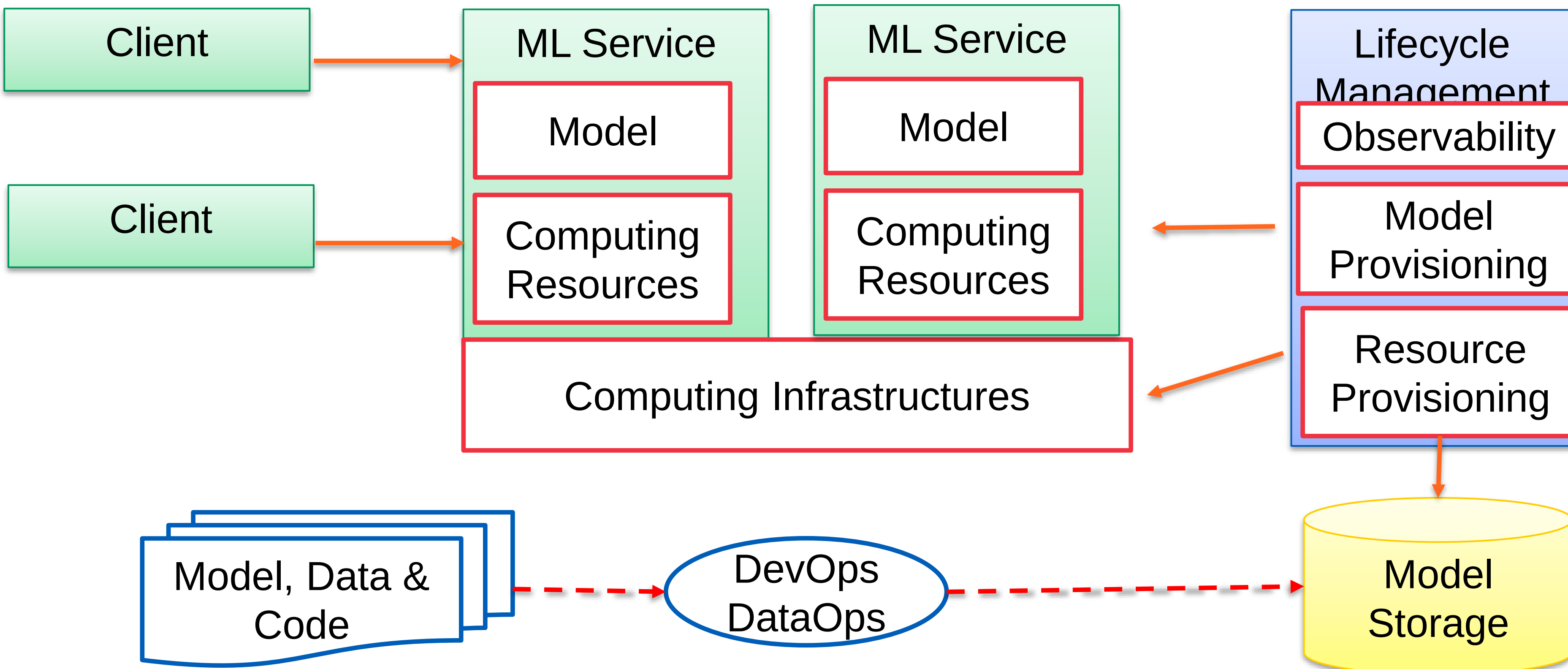
Figure source: Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: a Serverless Framework for End-to-end ML Workflows. In Proceedings of the ACM Symposium on Cloud Computing (SoCC '19). DOI:https://doi.org/10.1145/3357223.3362711

Dynamic ML Serving

ML Model Serving

- **Allow different versions of ML models to be provisioned**
  - runtime deployment/provisioning of models
  - "model as code" or "model as a service" → can be deployed into a hosting environment
- **Why? Anything related to R3E?**
  - concurrent deployments with different SLAs
  - A/B testing and continuous delivery for ML (https://martinfowler.com/articles/cd4ml.html)
- **Existing platforms**
  - increasingly support by different vendors as a concept of "AI as a service" (check https://github.com/EthicalML/awesome-production-machine-learning#model-deployment-and-orchestration-frameworks)

ML Model Serving design



ML Service

- **Long runtime inferencing services**
  - with well defined interfaces, know how to invoke models
  - accept continuous requests and serve in near-real time
- **Containerized service with REST/gRPC**
  - for on-demand serving or for scaling long running serving
- **Serverless function wrapping models**
  - short serving time
- **Batch serving**
  - not near real time serving due to the long inferencing time

Question: which forms are the best for which situations

Example: TensorFlow Extended Serving

- **Lifecycle**
  - Load, serving and unloading
- **Metrics & Policies**

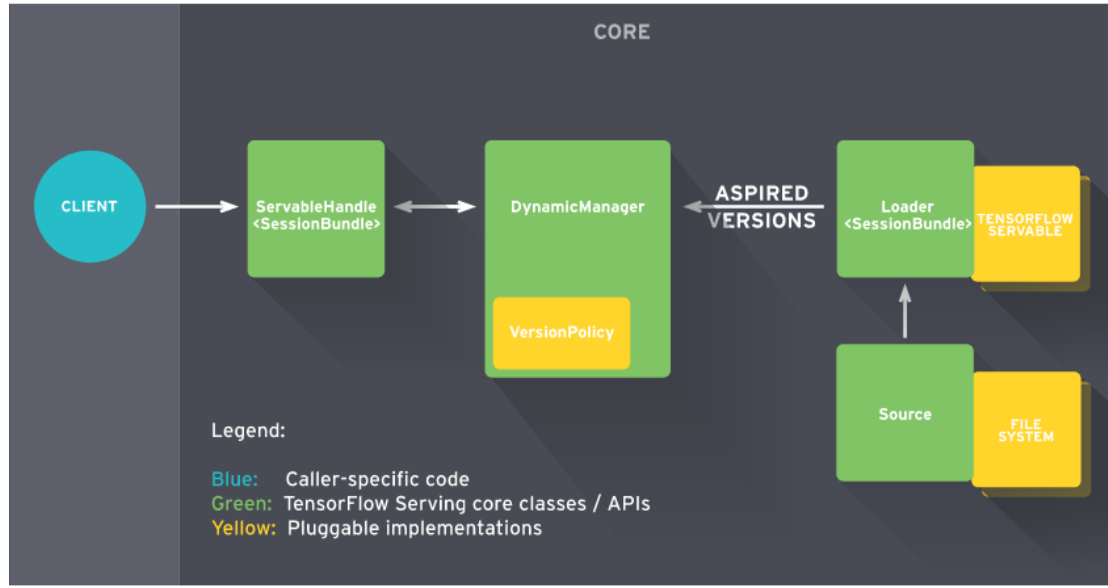
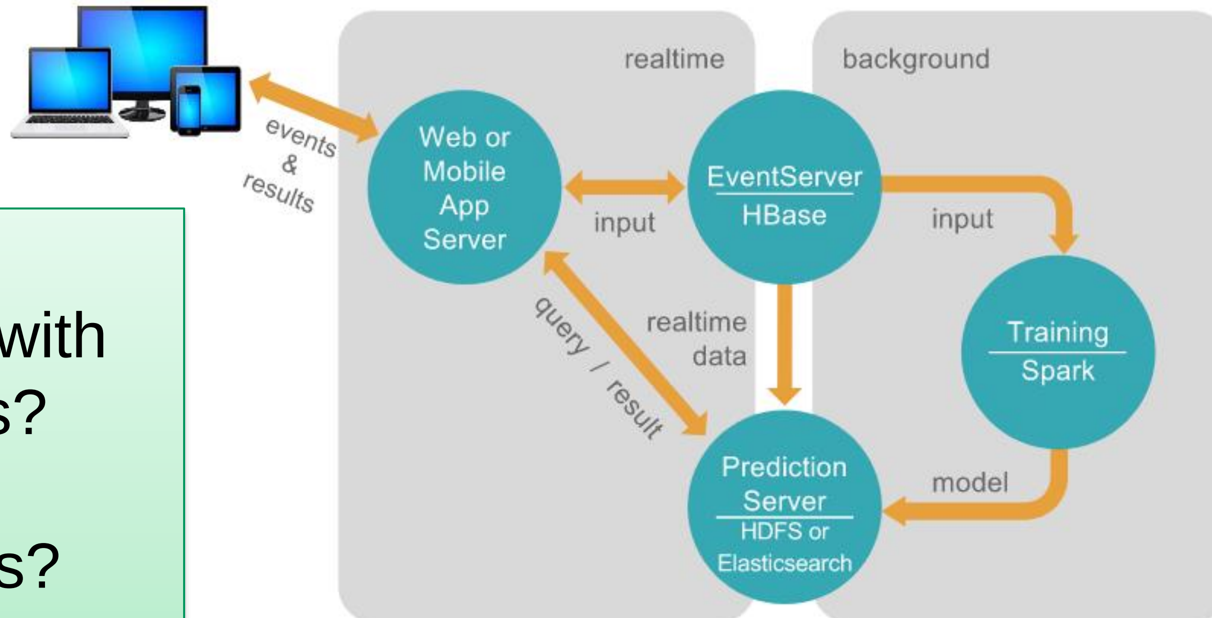


Figure source: https://www.tensorflow.org/tfx/serving/architecture

Example of Prediction.io

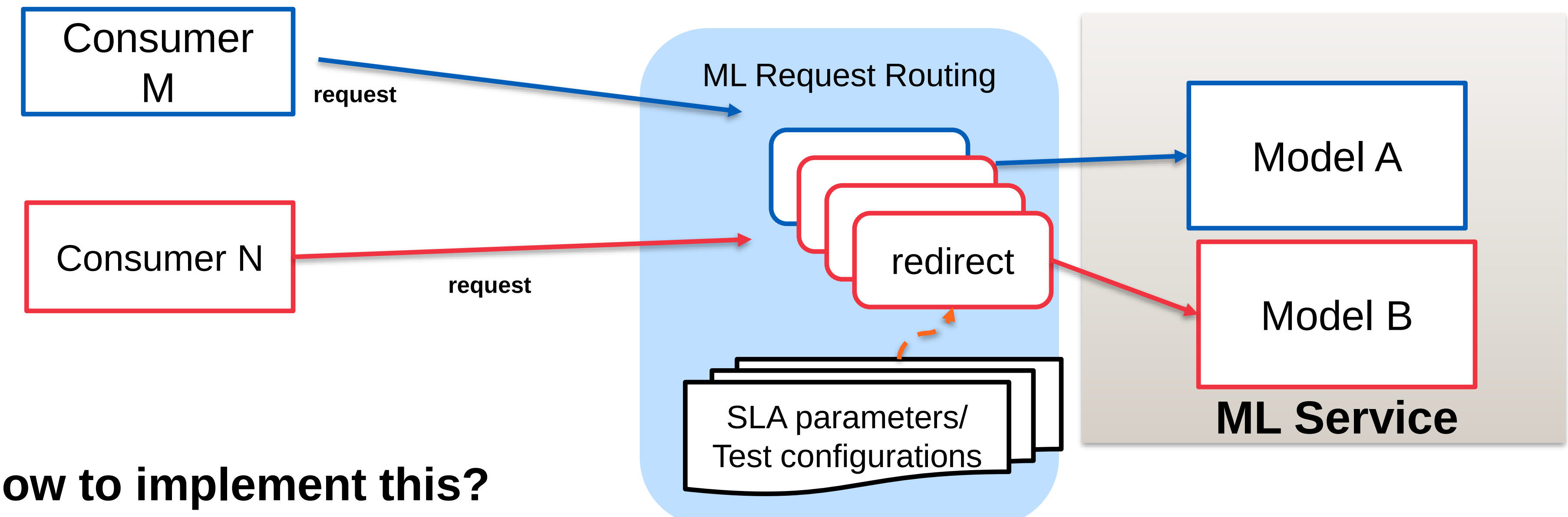


- Discussion: dealing R3E with ML workflows?
  - Elastic components?
  - Where, What, When and How

Figure source: https://predictionio.apache.org/system/

A/B testing and SLA-based serving

- **Different models with different qualities/SLAs**



How to implement this?

Example in Amazon Sagemaker

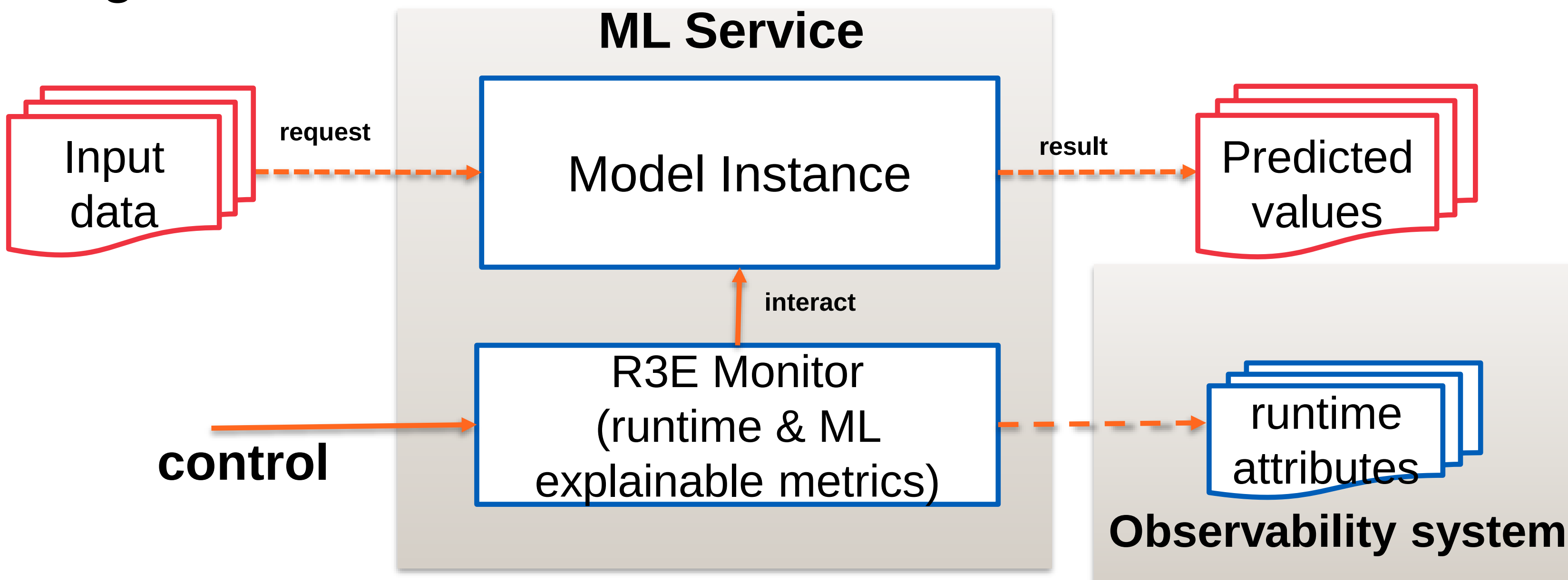
https://docs.aws.amazon.com/sagemaker/latest/dg/model-ab-testing.html

Load balancing/scaling model serving

- **ML inferencing capability in a ML model is encapsulated into a microservice or a task**
- **As a service**
  - with well-defined APIs (e.g., REST, gRPC), e.g., Dockerized service
  - using load balancing and orchestration techniques, such as Kubernetes
- **As a task**
  - using workflow management techniques to trigger new tasks
  - support scheduling, failure management and performance optimization by leveraging batch processing techniques

R3E Runtime attributes?

How to capture important metrics for observability and dynamic serving?



Aalto University
School of Science

What if a model is too big, need a lot of computing resources?

Study log

P1 - Take one of the following aspects:

- P1.1 - Robustness, Reliability, Resilience or Elasticity
- P1.2 – Automation management

P2 - Check one of the following aspects:

- Orchestration or ML model serving

In a *specific software framework* (F3) that you find interesting/relevant to your work:

discuss how do you see F3 supports P1 in doing P2

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io